Asgn3 Design

1 Introduction

Loadbalancer is a fault-tolerant multi-threaded application that will distribute http requests over several connected servers.

2 Data Design

I define a constant BUFFER_SIZE to be 4096 for simplicity because this is the maximum size a request header can be.

I then use a serverObject struct to contain relevant information for each connected server as follows.

- Int id
- Int port
- Int entries
- Int errors
- Bool alive

I also create a health_thread_Object to pass into my handle_task thread function

- a pointer to an array of servers
- integer of the number of servers
- pointer to the loadbalancers internal number of requests

3 Component Design

3.1 Main

The job of main will be to accept new connections, choose the next server then give the two file descriptors to a thread handle_task that will call bridge_loop on the two file descriptors until both are done. Main will also be responsible to perform a healthcheck on all servers every R requests

3.1.1 Choose_server

Choose server simply takes in an array of server objects and will loop the array checking the entries of each one to find the minimum. If there is a tie between two servers entries, it will choose the server with fewer errors. If it detects that all servers are down, it will return -1

3.1.2 Bridge_Connections

Bridge connections will simply receive BUFFER_SIZE bytes from one fd, place a null-terminating byte at the end, and send it to another fd. It will return the number of bytes sent. If the fd is closed, it will return 0. If there is an error on recv or send, it will return -1;

3.1.3 Bridge_loop

Bridge loop takes two sockets and will go into a loop that selects which one should be sending and which one should be receiving, then call bridge_connections until it returns <= 0.

3.1.3 Handle_task

Because p_thread create requires a single function that takes and returns a void pointer, I create a function 'handle_task' that will deal with each http request. Handle_task takes a pointer to a worker thread as an argument. It will first create an httpObject, then enter a while(true) loop, wait to be signaled by the dispatcher thread, then read, process and construct a response given a client socket, increment the entries/errors, write to the log if logging is enabled, and mark the worker as available.

3.2 Health checking

To perform healthchecks every X seconds or R requests, whichever comes first, I create an independent thread that will go into a while(true) loop that sleeps for X seconds, then if R requests have been made, continue. Otherwise it will loop through the array of servers and send a health_check_probe to each server

3.2.1 Health check probe

Health check probe simply sends a healthcheck request to the given server and will set the server objects entries, errors and alive status accordingly.

3.3 Multi-threading

To increase the throughput of a loadbalancer, it is advantageous to use threads. In my final implementation, I used a method not according to spec, but that is far easier to implement. In the main while loop, when a client is accepted and the correct available server is found, I simply call p_thread create to create a new thread that will bridge the file descriptors. After the thread is done, it will destruct. This is not ideal because it creates a lot of overhead to create a new thread for each request, but is easier to implement because I do not have to do any error checking if there are more requests than available threads.